THE AMERICAN UNIVERSITY IN CAIRO
الجامعة الأمريكية بالقاهرة

26/02/2020

Boot Loader Project

Phase 1 Final Report

Submitted to Dr. Karim Sobh

Submitted by:

Menna Elzahar	900182968
Omar Elsayed	900183884
Youssef Hussien	900183162
Nourhan Moussa	900172701

Phase_1 Recap

The first stage: completed the stage with code and comments needed, the stage has all the routines needed to print messages , to jump to second stage when done, it also have stored the second and third stage of boot loader in a certain segment address till when they are called to start loading

Second stage: completed the stage with code and comments needed, it will print a message that indicates that this stage is working and do three main things: check the status of A20 gate and enable it, check long mode support and finally scan the memory and store its regions' types data.

Work Distribution

Menna Elzahar: first stage including the partition table.

Omar Elsayed: second stage, created project on github

Youssef Hussien: second stage, report.

Nourhan Moussa: first stage, report

Assumptions

1. We had to reduce the content of some msgs to force the file to still be 512 bytes without times having a negative parameter. This is after replacing 510 with 446.
2. We know we had to check the carry flag after issuing the interrupt in load_boot_drive_params.asm to print error msg if an error happened, and then hang. However, after adding this part, we had to remove it to force the code to fit in the 446, especially when we know that probably there will be no error this time.

Findings

1- Reduce the messages in order to reduce the negatives

2- Partition Table

To implement the partition table, two steps were followed. First, 510 – ($-$$) db 0 was replaced with 446 – ($-$$) db 0 so that any code written after that line is included in the region following the boat loader code before the signature 2 bytes, 0x55 and 0xAA.

Second, the first table entry was defined so that

```
db 0x80              ;Boot/Active indicator
db 0x0               ;Starting Head Number
dw 0x0001            ;Starting Sector (1 in lower 6 bits) and starting cylinder (0 higher 10 bits)
db 'ext4'            ;System ID
db 0xFF              ;Ending Head Number 255
```

```
    dw 0xFFFF           ;Ending Sector (63 in lower 6 bits) and ending cylinder (1023 in higher 10
bits)
    dd 0x0              ;LBA Relative Sector Number is 0 (the MBR sector)
    dd 0x3F             ;Total number of sector is 63
For the next 3 sectors:
    db 0x80             ;Boot/Active indicator
    db 0x0              ;Starting Head Number
    dw 0x0000           ;Starting Sector (lower 6 bits) and starting cylinder (higher 10 bits)
    db 'ext4'           ;System ID
    db 0x00             ;Ending Head Number
    dw 0x0000           ;Ending Sector (lower 6 bits) and ending cylinder (higher 10 bits)
    dd 0x0              ;LBA Relative Sector Number is 0 (MBR sector)
    dd 0x0              ;Total number of sector
```

Steps Needed to Run the Code

1- We worked on the first stage code and edited the following files:

i) a.first_stage_data: a file that includes all memory variables and data definitions needed

ii) b. first_stage_main: this is the mail file that gathers all the functions and routines for this stage. As it starts with Macros for second and third stage with their address and when to start loading them. then it goes to main program that is responsible for calling routines to detect the disk and print messages that indicates that the program is working

iii) c. bios_print.asm: a subroutine that prints a string on the screen when needed, just in order to make sure that the program is working.

iv) d.read_disk_sectors.asm: a routine responsible for reading disk sector where it stored in the disk, they read 512 sectors.

v) e. load_boot_drive_params.asm: a subroutine that reads the parameters booted from the disk

vi) f. get_key_stroke.asm: a routine that allows the user to jump to second stage when he press the keyboard

vii) g. detect_boot_disk.asm: a subroutine to detect the parameters of a disk booted from

2. We worked on the second stage and edited/created the following files:

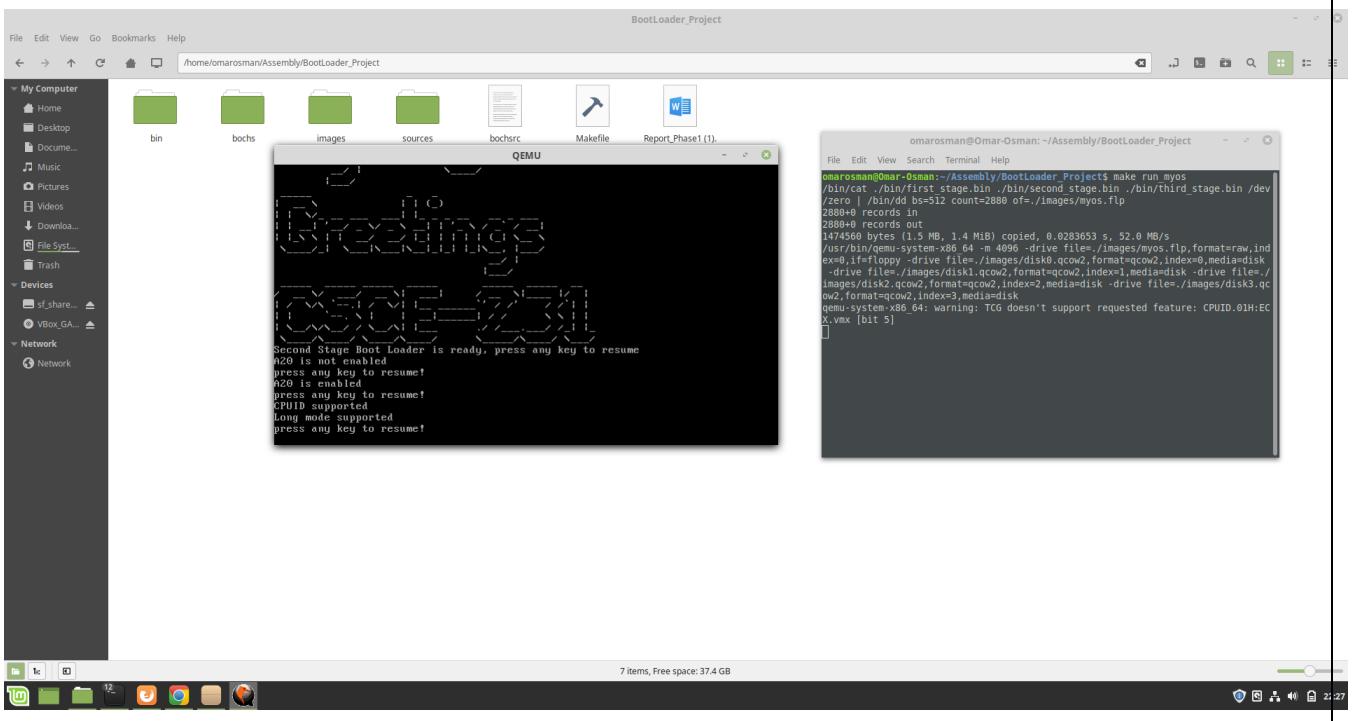
i) a20_gate.asm: This had two parts checing if A20 gate is supported and second is to check if it is enabled. To chec if the A20 gate is supported we used int 0x15 function 0x2402, which tells us that A20 is supported through setting Al, otherwise if it is zero then A20 is not

supported. Then we enabled the A20 gate through int 0x15 function 0x2401, which again sets Al if A20 is enabled , if it is zero then it is disabled.

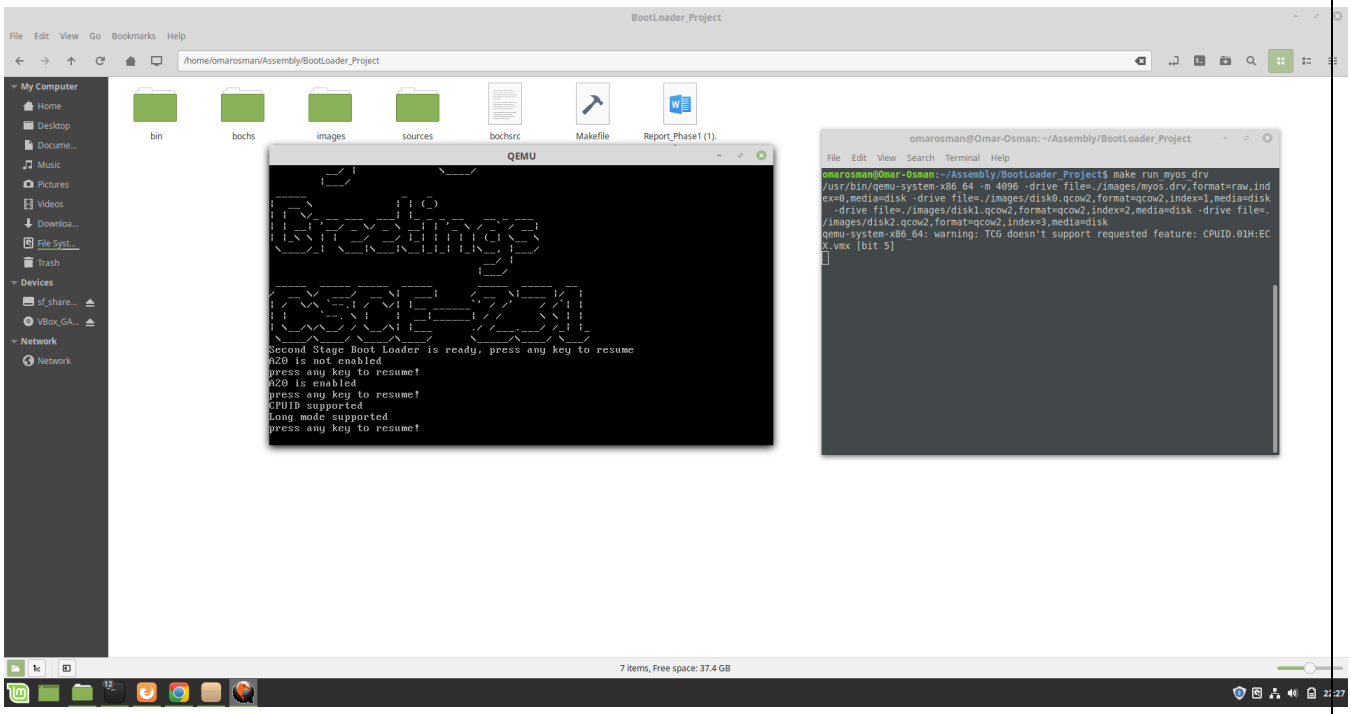
ii) check_long_mode.asm: To check the long mode support we needed to use the CPUID instruction which is supoorted only in the real mode, so we needed to use it in the real mode before transitioning to other modes. To chek the support of CPUID instruction we have to check if bit 21 in eflags is modifiable, if so then it is supported. Then we used funtion 0x80000001, and then check bit 29 if it is set, if yes then long-mode is supoorted.

iii) memory_scanner.asm: In this file we used function 0xe820 int 0x15 to read memory regions data and get the information of each region and stor them in the third segment of the memory. Moreover, we completed the memory_scan_failed label which prints out that memory scan is failed and jumps to hng.

Screen shot from floppy



Screen shot from drive


BootLoader_Project
File Edit View Go Bookmarks Help
/home/omarosman/Assembly/BootLoader_Project
My Computer
Home
Desktop
Music
Pictures
Videos
Trash
Devices
Network
bin
bochs
images
sources
bochsrc
Makefile
Report_Phase1 (1).
QEMU
Second Stage Boot Loader is ready, press any key to resume
A20 is not enabled
press any key to resume!
A20 is enabled
press any key to resume!
CPUID supported
Long mode supported
press any key to resume!
omarosman@Omar-Osman: ~/Assembly/BootLoader_Project
File Edit View Search Terminal Help
omarosman@Omar-Osman:~/Assembly/BootLoader_Project$ make run_myos_drv
/usr/bin/qemu-system-x86_64 -m 4096 -drive file=./images/myos.drv,format=raw,index=0,media=disk -drive file=./images/disk0.qcow2,format=qcow2,index=1,media=disk -drive file=./images/disk1.qcow2,format=qcow2,index=2,media=disk -drive file=./images/disk2.qcow2,format=qcow2,index=3,media=disk
qemu-system-x86_64: warning: TCG doesn't support requested feature: CPUID.01H:ECX.vmx [bit 5]
7 items, Free space: 37.4 GB